Contents

JWT Token

4 minute read ✔ page test

This task shows you how to set up an Istio authorization policy to enforce access based on a JSON Web Token (JWT). An Istio authorization policy supports both string typed and list-of-string typed JWT claims.

Before you begin

Before you begin this task, do the following:

- Complete the Istio end user authentication task.
- Read the Istio authorization concepts.
- Install Istio using Istio installation guide.
- Deploy two workloads: `httpbin` and `sleep`. Deploy these in one namespace, for example `foo`. Both workloads run with an Envoy proxy in front of each. Deploy the example namespace and workloads using these commands:

```
$ kubectl create ns foo
$ kubectl apply -f <(istioctl kube-inject -f @samples/httpb
in/httpbin.yaml@) -n foo
$ kubectl apply -f <(istioctl kube-inject -f @samples/sleep
/sleep.yaml@) -n foo
```

- Verify that `sleep` successfully communicates with `httpbin` using this command:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o js
onpath={.items..metadata.name})" -c sleep -n foo -- curl ht
tp://httpbin.foo:8000/ip -sS -o /dev/null -w "%{http_code}\
n"
200
```

If you don’t see the expected output, retry

after a few seconds. Caching and propagation can cause a delay.

Allow requests with valid JWT and list-typed claims

1. The following command creates the `jwt-example` request authentication policy for the `httpbin` workload in the `foo` namespace. This policy for `httpbin` workload accepts a JWT issued by

`testing@secure.istio.io`:

```
$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: "jwt-example"
  namespace: foo
spec:
  selector:
    matchLabels:
      app: httpbin
  jwtRules:
  - issuer: "testing@secure.istio.io"
    jwksUri: "https://raw.githubusercontent.com/istio/istio
/release-1.11/security/tools/jwt/samples/jwks.json"
EOF
```

2. Verify that a request with an invalid JWT is

denied:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o js
onpath={.items..metadata.name})" -c sleep -n foo -- curl "h
ttp://httpbin.foo:8000/headers" -sS -o /dev/null -H "Author
ization: Bearer invalidToken" -w "%{http_code}\n"
401
```

3. Verify that a request without a JWT is allowed because there is no authorization policy:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o js
onpath={.items..metadata.name})" -c sleep -n foo -- curl "h
ttp://httpbin.foo:8000/headers" -sS -o /dev/null -w "%{http
_code}\n"
200
```

4. The following command creates the `require-jwt`

authorization policy for the `httpbin` workload in the `foo` namespace. The policy requires all requests to the `httpbin` workload to have a valid JWT with `requestPrincipal` set to `testing@secure.istio.io/testing@secure.istio.io`. Istio constructs the `requestPrincipal` by combining the `iss` and `sub` of the JWT token with a `/` separator as shown:

```
$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: require-jwt
  namespace: foo
spec:
  selector:
    matchLabels:
      app: httpbin
  action: ALLOW
  rules:
  - from:
    - source:
       requestPrincipals: ["testing@secure.istio.io/testing
@secure.istio.io"]
EOF
```

5. Get the JWT that sets the `iss` and `sub` keys to the

same value, `testing@secure.istio.io`. This causes Istio to generate the attribute `requestPrincipal` with the value `testing@secure.istio.io/testing@secure.istio.io`:

```
$ TOKEN=$(curl https://raw.githubusercontent.com/istio/istio/release-1.11/security/tools/jwt/samples/demo.jwt -s) && echo "$TOKEN" | cut -d '.' -f2 - | base64 --decode -
{"exp":4685989700,"foo":"bar","iat":1532389700,"iss":"testing@secure.istio.io","sub":"testing@secure.istio.io"}
```

6. Verify that a request with a valid JWT is allowed:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o js
onpath={.items..metadata.name})" -c sleep -n foo -- curl "h
ttp://httpbin.foo:8000/headers" -sS -o /dev/null -H "Author
ization: Bearer $TOKEN" -w "%{http_code}\n"
200
```

7. Verify that a request without a JWT is denied:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o js
onpath={.items..metadata.name})" -c sleep -n foo -- curl "h
ttp://httpbin.foo:8000/headers" -sS -o /dev/null -w "%{http
_code}\n"
403
```

8. The following command updates the `require-jwt` authorization policy to also require the JWT to have a claim named `groups` containing the value

```
group1:
```

```
$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: require-jwt
  namespace: foo
spec:
  selector:
    matchLabels:
      app: httpbin
  action: ALLOW
  rules:
  - from:
    - source:
       requestPrincipals: ["testing@secure.istio.io/testing
@secure.istio.io"]
    when:
    - key: request.auth.claims[groups]
      values: ["group1"]
EOF
```

Don’t include quotes in the `request.auth.claims` field unless the claim itself has quotes in it.

9. Get the JWT that sets the `groups` claim to a list of strings: `group1` and `group2`:

```
$ TOKEN_GROUP=$(curl https://raw.githubusercontent.com/istio/istio/release-1.11/security/tools/jwt/samples/groups-scope.jwt -s) && echo "$TOKEN_GROUP" | cut -d '.' -f2 - | base64 --decode -
{"exp":3537391104,"groups":["group1","group2"],"iat":1537391104,"iss":"testing@secure.istio.io","scope":["scope1","scope2"],"sub":"testing@secure.istio.io"}
```

10. Verify that a request with the JWT that includes `group1` in the `groups` claim is allowed:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o js
onpath={.items..metadata.name})" -c sleep -n foo -- curl "h
ttp://httpbin.foo:8000/headers" -sS -o /dev/null -H "Author
ization: Bearer $TOKEN_GROUP" -w "%{http_code}\n"
200
```

11. Verify that a request with a JWT, which doesn't have the `groups` claim is rejected:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o js
onpath={.items..metadata.name})" -c sleep -n foo -- curl "h
ttp://httpbin.foo:8000/headers" -sS -o /dev/null -H "Author
ization: Bearer $TOKEN" -w "%{http_code}\n"
403
```

Clean up

1. Remove the namespace `foo`:

```
$ kubectl delete namespace foo
```